KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Ville-Pekka Pouhula

MOBIILI KÄYTTÖLIITTYMÄ MOVENIUM TIMETRACKER – TYÖAJANSEURANTAPALVELUUN

Opinnäytetyö
Maaliskuu 2016

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Maaliskuu 2016
Tietotekniikan koulutusohjelma

Karjalankatu 3
80200 JOENSUU
013 260 600

Tekijä(t)
Ville-Pekka Pouhula

Nimeke
Mobiili käyttöliittymä Movenium TimeTracker -työajanseurantapalveluun

Toimeksiantaja
Movenium Oy

Tiivistelmä

Tämän työn tarkoituksena oli suunnitella ja kehittää uusi käyttöliittymä Movenium TimeTracker -työajanseurantapalveluun. Käyttöliittymä suunniteltiin toimimaan pääasiassa älylaitteissa. Opinnäytetyössä on käsitelty eri ohjelmistokehitystekniikoita ja työkaluja. Työn loppuosiossa on testattu uutta kehitystekniikkaa nopeustestien avulla.

Opinnäytetyö toteutettiin toimeksiantona espoolaiselle ohjelmistotalo Movenium Oy:lle. Tarve uudelle sovellukselle oli suuri, koska toimeksiantajalla ei ollut kunnollista sovellusta älylaitteille. Tämä opinnäytetyö toimi esimerkkinä Movenium Oy:n seuraaville työajanseurantapalvelun versioille. Työssä käytettiin toimeksiantajan tarjoamia rajapintoja ja palveluja.

Opinnäytetyö oli pääasiassa toiminnallinen, mutta sisältää myös jonkin verran tutkimusta eri tekniikoista. Opinnäytetyö aloitettiin vuonna 2013 ja varsinainen toiminnallinen osuus saatiin päätökseen melko nopeasti. Tämän työn pohjalta toimeksiantaja on kehittänyt uutta versiota, joka tällä hetkellä (2016) on jo tuotantokäytössä ja useat asiakkaat käyttävät sitä.

Kieli
suomi

Sivuja 42

Asiasanat
web-sovellus, älylaitteet, JavaScript, Ember.js

THESIS
March 2016
Degree Programme in Information Techonolgy
Karjalankatu 3
FI 80200 JOENSUU
FINLAND
013 260 600

Author(s)
Ville-Pekka Pouhula

Title
Mobile interface for Movenium TimeTracker

Commissioned by
Movenium Oy

Abstract

Main goal for this thesis was to design and implement a new mobile user interface for a time tracking service called Movenium TimeTracker. Primary end-user device for the new application was smart devices. There was comparison of different programming techniques and tools in this thesis. The new and old techniques are being compared to each other at the conclusion.

Thesis was made as an assignment for software company called Movenium located in Espoo. Movenium had a dire need for a new application which should work in smart devices. This thesis was also a foundation for future versions of time tracking software. Movenium offered interfaces and services for the time tracking service.

Thesis was chiefly a functional work but it also includes slightly a research of some different techniques. Starting point of this thesis was at 2013 and the implementing part was done quickly. Movenium has developed a new version of the time tracking service based on this thesis. This new version is now in production use and many customers are already using it.

Language
Finnish

Pages 42

Keywords

web application, smart device, JavaScript, Ember.js

Sisältö

1 Johdanto

Tässä opinnäytetyössä suunniteltiin ja toteutettiin mobiili käyttöliittymä Movenium TimeTracker -työajanseurantapalveluun. Opinäytetyö suoritettiin toimeksiantona Movenium Oy:lle ja sen tarkoituksena on toimia esimerkkinä tuleville sovellusratkaisuille.

Älylaitteiden osuus internetin käyttämisessä on kasvanut todella paljon viime vuosina. Vuonna 2013 älypuhelimia oli jopa enemmän kuin tavallisia puhelimia [1] ja tämä asettaa tiettyjä haasteita niin käytettävyyteen kuin teknisiin ratkaisuihin. Älylaitteissa on useimmiten kosketusnäyttö ja tiedonsiirto tapahtuu langattomia verkkoja pitkin.

On selkeä trendi, että älylaitteiden osuus on kasvamassa koko ajan internetin käytössä [2]. Sovellusten käyttöliittymät täytyy tämän takia suunnitella erityisesti näitä päätelaitteita ajatellen. Tässä työssä suunnitellaan sovellus pääasiassa älylaitekäyttöä ajatellen. Kun sovellus toimii älylaitteilla, se on helppo sovittaa työpöytänäkymään.

Normaalisti web-sovelluksessa jokaisella sivulatauksella ladataan paljon ja usein toistuvaa dataa, joka vaatii ylimääräistä käsittelyä palvelimelta ja se vie myös enemmän kaistaa. Uusia tekniikoita hyödyntäen voidaan kuitenkin sivuston eri osioita ladata osissa sen mukaan, miten loppukäyttäjä niitä tarvitsee.

Tässä työssä oli tarkoitus tutkia näitä uusia tekniikoita, joita voidaan hyödyntää älylaitteiden web-sovelluksissa. Uusien tekniikoiden on tarkoitus parantaa loppukäyttäjän käyttökokemusta, keventää palvelinkuormaa ja parantaa tietoturvaa.

Työn teoriaosuudessa tutkittiin eri ohjelmointitekniikoiden hyötyjä ja periaatteitta ja toteutusosiossa perehdyttiin niiden soveltuvuuteen esimerkkityöhön.

2 Opinnäytetyön lähtökohdat

2.1 Web-sovellus

Web-sovellus on yksinkertaisesti ohjelma, joka käyttää hyödykseen palvelin-asiakasmallia. Web-sovelluksien jakelu tapahtuu internetin kautta ja vastaanottava rajapinta on yleensä HTML-selain. Verkkoselain, eli asiakasohjelma, näyttää käyttöliittymän, jossa loppukäyttäjä käyttää sovellusta. Palvelin säilyttää suurimman osan sovellukseen liittyvästä datasta ja huolehtii asiakkaan tekemistä toiminnoista. [3]

Työn yhtenä lähtökohtana oli selvittää, tuovatko paljon suosiota saaneet JavaScript-sovelluskehykset helpotusta ohjelmistojen kehittämiseen ja parantavatko ne loppukäyttäjän kokemusta sovelluksen parissa.

Perinteinen web-sovellus toimii niin, että koko sivun sisältö ladataan aina joka sivulatauksella. Selaimissa on välimuisti, joka auttaa esimerkiksi kuvien ja tyylitiedostojen osalta; näitä ei tarvitse ladata joka kerta. Kun dynaamiseksi tarkoitettu sisältö päivittyy, niin silloin täytyy päivittää myös koko html-dokumentti, jotta saadaan dynaamisesti muuttunut sisältö esitettyä loppukäyttäjälle. Yleensä sivusto on samalla palvelimella myös muiden ohjelmiston osioiden, kuten tietokannan kanssa, eli palvelinresurssit joudutaan jakamaan sovelluksen eri komponenttien kesken.

JavaScript-sovelluskehyksen avulla toteutetun sovelluksen ajamiseen tarvittavat tiedostot ladataan ensin selaimen muistiin palvelimelta ja tämän jälkeen sovellus toimii jopa ilman verkkoyhteyttä. Tietoa pitää kuitenkin edelleen hakea dynaamisesti palvelimelta rajapintojen kautta. Sovelluskehykset on suunniteltu niin, että tieto haetaan ja käsitellään asynkronisesti sovelluksen sisällä, eikä koko sivua tarvitse aina ladata uudestaan.

2.2 Sovelluksen tyyppi älylaitteessa

Aikaisempina vuosina matkapuhelimien verkkoselaimet ovat olleet melko yksinkertaisia ja tuki erilaisille tekniikoille (esim. JavaScript) on ollut puutteellista. Nykyään kuitenkin älylaitteiden selaimet tukevat esimerkiksi HTML5:ttä melkein poikkeuksetta [4]. Natiivia sovelluksia ei siis tarvitse enää käyttää ja sovellus voidaan ajaa verkkoselaimessa web-sovelluksena.

Suurin osa älylaitteiden sovelluksista käyttää verkkoyhteyttä ja rajapintoja tiedon tuomiseen päätelaitteelle loppukäyttäjälle sopivaan muotoon. Otetaan esimerkiksi vaikka sovellus, joka näyttää sanomalehden uutisia; sovellus hakee uusimmat uutiset uutispalvelun rajapinnan kautta ja näyttää ne päätelaitteessa sopivassa muodossa. Tämänlainen sovellus on hyvin helppo tehdä web-sovelluksena, jolloin natiivin sovelluksen tuomat hyödyt jäävät hieman kyseenalaiseksi.

Natiivit sovellukset sopivat paremmin sellaiseen tehtävään, jossa tarvitaan älylaitteen resursseja, joihin verkkoselaimella ei ole pääsyä, esimerkiksi puhelimen osoitekirjaan tai NFC-rajapintaan.

Riippuu siis hyvinkin paljon sovelluksen toiminnallisuudesta, kannattaako se tehdä web-sovelluksena vai natiivina sovelluksena. On myös olemassa erilaisia työkaluja, joilla web-sovellus voidaan kääntää natiiviksi sovellukseksi suhteellisen helposti ja mukaan saadaan vielä paljon erilaisia rajapintoja puhelimen resursseihin. Tästä yksi hyvä esimerkki on Apache Cordova [5], joka on suosittu työkalu web-sovellusten kääntämiseen eri puhelinalustoille.

2.3 Web-sovellusten erilaiset sovellusalustat

Web-sovelluksen suoritusohjelmana toimii aina verkkoselain. Verkkoselaimia on useita erilaisia. Älylaitteissa ja tietokoneissa on usein saman valmistajan selaimia, mutta näissä on suuria rakenteellisia eroja.

Yksi suuri ongelma web-sovelluksissa on se, että ne ovat alustariippumattomia. Tämä voi vaikuttaa hyvältä asialta, mutta tosiasia on se, että sovelluksen suoritusalustana toimivat selaimet käsittelevät asioita jokainen hieman omalla tavallaan. Vaikka HTML:lle ja JavaScriptille on kehitetty standardit, joita selainvalmistajat pyrkivät noudattamaan, niin silti jotain eroavaisuuksia löytyy aina. Onkin tärkeää, että sovelluksen kehitysvaiheessa pyritään käyttämään oikeaoppisia ratkaisuja ohjelmoinnissa ja testaamaan sovellusta kehityksen aikana eri alustoilla. Lopputestauksen aikana tulleet ongelmat voivat olla jo siinä vaiheessa melko hankalia ratkaista.

Oma haasteensa tulee myös työpöytäselaimien ja älylaitteiden selaimien erilaisista ohjelmistokoodin tulkintatavoista. Selaimien sisäiset komponentit voivat olla aivan erinäköisiä eri selaimien välillä. Esimerkiksi JavaScriptin *alert()*-funktio voi näyttää tietokoneella aivan erilaiselta kuin matkapuhelimella.

2.4 Työajanseuranta ja tietojen kerääminen

Suomen laissa on määritelty työaikalaki [6], joka velvoittaa työnantajan kirjaamaan työhön liittyvät työtunnit. Nykyään tietotekniikka auttaa paljon työaikojen kirjaamisessa ja on olemassa monia työajanseurantajärjestelmiä. Työajanseuranta tarkoittaa käytännössä sitä, että työntekijä kirjaa työpäivän tunnit ylös. Kirjattavina tietoina voivat olla joko työn aloitus- ja lopetusaika tai tehdyt tunnit, sekä työn tyyppi ja työskentelypaikka.

Työajanseurantajärjestelmissä seurataan myös usein muutakin tietoa, kuten poissaoloja ja työhön liittyviä matkoja. Työajanseuranta usein myös helpottaa palkanlaskentaa ja laskutusta, koska tiedot löytyvät helposti sähköisessä muodossa.

2.5 Movenium TimeTracker -työajanseurantapalvelu

Tässä työssä taustapalveluna toimii suomalaisen ohjelmistoyritys Moveniumin TimeTracker -työajanseurantapalvelu. Movenium on vuonna 2007 perustettu

ohjelmistotalo, joka toimii pääasiassa Suomessa mutta toimintaa on myös Ruotsissa. TimeTracker on ns. SaaS-palvelu [7], eli palvelua tarjotaan käytettäväksi internetin kautta. Asiakas ei näin ollen tarvitse minkäänlaisia asennuksia omalle tietokoneelleen ja palvelu toimii kaikissa laitteissa, joissa on verkkoyhteys ja -selain. SaaS-palvelu huolehtii itsestään ohjelmiston päivityksistä ja varmuuskopioista.

TimeTracker on helppokäyttöinen ja hyvin muokattavissa asiakkaiden tarpeiden mukaan. TimeTracker toimii vanhoilla ja uusilla puhelinmalleilla sekä tabletilla ja tietenkin tietokoneella. TimeTrackeriin on myös saatavilla erillinen leimauslaite, joka toimii NFC-tekniikalla. Tämän avulla työntekijät eivät tarvitse kuin RFID-avaimenperän työaikojen leimaamiseen. TimeTrackerilla on noin 20 000 päivittäin työaikaansa merkitsevää työntekijää ja noin 1100 asiakasyritystä. Suurin osa asiakkaista on rakennusalan yrityksiä. [8]

3 Tekniikat ja työkalut

3.1 JavaScript

JavaScript on oliopohjainen kevytrakenteinen ohjelmointikieli, joka on tarkoitettu helpottamaan web-sovellusten kehittämistä. JavaScriptiä käytetään eritoten helpottamaan käyttöliittymän toimintoja vuorovaikutteisuutta lisäämällä. JavaScript on prototyyppipohjainen dynaaminen skriptikieli, joka tulkataan yleensä loppukäyttäjän selaimessa. [9]

JavaScript oli alkuaikoinaan 90-luvulla todella halveksittu työkalu web-sovellusten kehittäjien keskuudessa. Yleensä suositeltiin jopa ottamaan JavaScript kokonaan pois päältä selainasetuksista. Sen ominaisuuksia käytettiin enemmän käyttäjän kiusaamiseen kuin sen palvelemiseen. Myöhemmin kuitenkin erilaiset kehityskirjastot yleistyivät ja kehittäjät oppivat käyttämään JavaScriptiä siihen mihin se oli tehokas; käyttöliittymän ja käyttökokemuksen parantamiseen. [35]

JavaScriptillä ei ole kuitenkaan nimestään huolimatta mitään tekemistä Java-ohjelmointikielen kanssa. JavaScriptin kehitti 10 päivän aikana toukokuussa 1995 Brendan Eich. Nimi oli alun perin Mocha, joka muutettiin LiveScriptiksi ja viimein markkinointisyistä JavaScriptiksi. Eich työskenteli tuolloin Netscapelle ja nykyään Mozillalle. [10]

Nykyään puhdasta JavaScript-koodia harvemmin käytetään, sillä funktiokirjastot (jQuery, Dojo, MooTools) ovat vallanneet markkinoita. Nämä tuovat kehittäjille suurta hyötyä, koska usein käytetyt toiminnot on tehty järkevämmin. Esimerkiksi Ajaxin käyttö on hankalaa ilman erillistä kirjastoa.

On myös olemassa kokonaisia sovelluskehyksiä (Ember.js, AngularJS, Backbone.js, JavaScriptMVC), jotka ovat huomattavasti laajempia kokonaisuuksia kuin pelkät funktiokirjastot. Näissä on yleensä määritetty tietty tapa ohjelmointiin ja mukana on usein työkaluja testaukseen.

Työssä käytettyjä JavaScriptin kehityskirjastoja ja sovelluskehyksiä on tutkittu lisää myöhemmissä kappaleissa.

3.2 HTML

HyperText Markup Language eli HTML on internetin sivustoilla käytetty kuvauskieli, jolla voidaan kuvata web-sivuja. On olemassa muitakin merkkauskieliä, mutta HTML on näistä jäänyt oikeastaan ainoana voimaan web-sivujen merkkauskielenä. HTML ei ole ohjelmointikieli, kuten yleisesti luullaan, vaan merkkauskieli. Sen avulla kerrotaan, miltä sivun kuuluisi näyttää selaimessa ja käyttäjän näytöllä. [11, s. 15.]

Pelkkä HTML on oikeastaan jo vanhentunut tekniikka ja sivustoilla tulisi käyttää sen uudempaa versiota tai XHTML:ää. XHTML polveutuu XML:stä, joka on merkkauskieli HTML:n tapaan mutta on huomattavasti tarkempi ja vankempi sääntöjen suhteen. XHTML:ssä on kaikki HTML:n ominaisuudet, mutta se tukee

myös CSS:ää ja sisältää parempia ominaisuuksia web-sivujen kehitykseen. [11, s. 24.]

Uusin versio HTML:stä on versio numero 5, jonka tarkoituksena on tuoda web-sivujen kehitys helpommaksi myös mobiililaitteille. HTML5:n tarkoitus on olla universaali merkkauskieli, joka toimii kaikkialla. Siinä määritellään kuinka asiat esitetään ja kuinka selaimet ne ymmärtävät. HTML5:tä on kehitetty jo vuodesta 2004, mutta se ei ole vieläkään täysin valmis. Tästä huolimatta HTML5:tä käytetään jo laajalti internetissä ja useat mobiililaitteet (iPhone, Android ja tabletit) tukevat jo nyt sitä, ei ehkä kokonaisuudessaan mutta jotain osioita. HTML5-tuki laitteissa kasvaa kuitenkin jatkuvasti. [11, s. 26.]

3.3 AJAX

AJAX on yhdistelmä erilaisia tekniikoita, joiden ansiosta sivustojen dynaamisuutta voi kehittää huomattavasti paremmaksi. AJAX tulee sanoista *Asynchronous JavaScript And XML*, joka alun perin tarkoitti sitä, että tarvittava data on XML-dokumentissa ja tämä voidaan ladata asynkronisesti HTTP-palvelimelta eri osoitteesta kuin itse sovellus. Nykyään kuitenkin käytetään useammin JSON-muotoista dataa ja Ajaxiin liittyy myös merkkauskielen (HTML) ja tyylitiedostojen (CSS) muokkaamista. [12, s. 1.]

Ajax on siis web-sovelluskehityksen tyyli, jossa voidaan dynaamisesti muokata sovelluksen käyttöliittymää käyttämällä ohjelmointikieltä, joka lataa vain tarvittavaa dataa palvelimelta. Pääpaino on sanoissa *dynaaminen* ja *vain tarvittaessa*. [12, s. 1.]

3.4 PHP

PHP eli “PHP: Hypertext Preprocessor” on hyvin laajalti käytetty ohjelmointikieli web-palvelinympäristöissä. PHP:ta käytetään usein luomaan dynaamista sisältöä sivustoille. PHP:n syntaksi nojautuu C:hen, Javaan ja Perliin, ja se on helposti opittavissa. PHP:n päätarkoitus on antaa ohjelmistokehittäjille työkalu, jolla

voidaan tehdä nopeasti dynaamisia web-sivustoja. PHP:lla voi tosin tehdä paljon muutakin. [13]

PHP:n ensimmäinen versio vuodelta 1995 oli nimeltään PHP/FI ja sitä kehitti Rasmus Lerdorf. Hän teki aluksi yksinkertaisen skriptin, joka seurasi hänen ansioluettelonsa katsojamääriä. Tuossa versiossa oli vain jotain yksinkertaisia funktioita ja tuki mSQL-tietokannalle. PHP:n 3-versiota kehittivät Andi Gutmans ja Zeev Suraski, jotka käytännössä kirjoittivat PHP:n uusiksi. Tämän version myötä PHP alkoi kasvattaa suosiotaan web-palvelinten skriptikielenä. [14, s. 8.]

3.5 JavaScript-sovelluskehykset

Erilaisia sovelluskehyksiä löytyy paljon ja näistä on hankalaa löytää itselleen juuri se sopiva. Jokainen on omalla tavallaan hyvä ja parasta on hankala, ellei jopa mahdotonta, määritellä. Tässä osiossa esitellään kolme suosittua sovelluskehystä ja tutkitaan hieman niiden hyviä ja huonoja puolia.

Työssäni päädyin valitsemaan Ember.js:n, koska se tukee laajojakin sovellusrakenteita ja sisältää oman tietovaraston datan säilömiseen. Ember.js:n iskulauseenakin on, että sillä voi tehdä kunnianhimoisia sovelluksia. Ember.js on nopeasti kehittyvä ja jatkuvasti tulee hyviä uusia ominaisuuksia, jotka auttavat ohjelmistokehittäjien työtä. Ember.js:n sosiaalinen verkosto on aktiivinen ja apua saa helposti ja nopeasti omiin ongelmiin.

Ember.js:n kehittämiseen käytetään Ember CLI-työkalua, joka sisältää useita tärkeitä komponentteja kuten esim. sovelluksen kehityksenaikaisen testaamisen, kolmansien osapuolien paketinhallinnan, sovelluksen kääntämisen ja sovelluksen julkaisemisen.

En kuitenkaan väitä, että Ember.js olisi mitenkään järkevin tai paras valinta sovelluskehykseksi; riippuu täysin projektista, mikä sovelluskehys sopii mihinkin tarkoitukseen parhaiten.

3.5.1 Ember.js

Ember.js on MVC-mallin mukainen front-end JavaScript-sovelluskehys, jota suoritetaan selaimessa. Ember.js on suunniteltu laajoille sovelluksille, joiden on tarkoitus olla vastine natiiveille sovelluksille, ja se on myös kehitetty näiden sovelluskehysten pohjalta, esimerkiksi Cocoasta. Ember.js auttaa kehittämään käyttäjälle hyvän käyttökokemuksen. Ember.js tulee hyödylliseksi silloin, kun normaali JavaScript-sovellus alkaa käydä turhan laajaksi ja sovellusarkkitehtuurin hallinta ei ole enää helppoa. [15, s. 1.]

Ember.js:n hyviä puolia ovat

- **Navigaatio.** Ember.js:n routet pitävät huolta sovelluksen navigaatiosta.
- **Automaattisesti päivittyvät mallit**. Kaikki sovelluksessa kulkeva data päivittyy aina automaattisesti mallit, kun sitä muokataan.
- **Datan käsittely.** Jokainen luotu objekti on Ember-objekti, johon periytetään kaikki Ember.object:n metodit.
- **Asynkroninen käyttäytyminen.** Ember.js:n ominaisuuksiin kuuluu tietojen haku asynkronisesti. [15, s 2.]

3.5.2 AngularJS

AngularJS on sovelluskehys, jonka tarkoitus on tarjota työkalut yksisivuisten web-sovellusten rakentamiseen. AngularJS on melko samanlainen kuin Ember.js ja käyttölogiikassa ei ole suuria eroja. [16]

3.5.3 Backbone.js

Backbone.js on myös MVC-mallin mukainen sovelluskehys, mutta on vieläkin yksinkertaisempi. Se helpottaa kehittäjiä ylläpitämään sovelluksen rakennetta yleisesti ja muistuttaa näistä eniten perinteistä JavaScriptiä. Backbone.js on myös helpoin omaksua yksinkertaisuutensa vuoksi. Yksinkertaisuudella on

myös huonot puolensa; sovelluksen arkkitehtuuri täytyy määritellä kokonaan itse, jonka vuoksi sovelluksesta voi helposti tulla sekava. [17]

3.6 Muita tekniikoita

Työssä käytettiin myös monia muita erilaisia tekniikoita ja ohjelmistoja. Nykypäivän ohjelmistokehittäminen vaatii osaamista monista eri työkaluista ja tekniikoista.

3.6.1 jQuery

jQuery on ehdottomasti suosituin JavaScriptille tehty kehityskirjasto [18]. Se sisältää lukuisia funktioita, jotka helpottavat HTML-dokumenttien muokkausta ja manipulaatiota, selaintapahtumien käsittelyä, animaatioita ja Ajaxin kanssa työskentelyä. jQuery on laajalti tuettu eri selaimissa ja se on helppokäyttöinen sekä hyvin dokumentoitu.

jQueryn yksi tärkeimmistä periaatteista on helpottaa kehittäjän työtä. jQuery antaa kehittäjälle käyttöön työkalut, joilla eri toimintojen toteuttaminen onnistuu huomattavasti helpommin ja vähäisemmällä koodirivien määrällä, kuin puhtaalla JavaScriptillä kirjoittamalla. [19]

3.6.2 Bootstrap

Bootstrap on kokoelma työkaluja ja komponentteja, joilla helpotetaan websivuston käyttöliittymän rakentamista. Bootstrap sisältää lähinnä HTML:llä ja CSS:llä rakennettuja valmiita rakenteita kuten kirjasinlajeja, ikoneita, lomakkeiden elementtejä, nappeja sekä navigaatio- ja käyttöliittymäkomponentteja. Bootstrap sisältää myös kehitystyötä helpottavia JavaScript-laajennuksia. [20]

3.6.3 REST

REST on suosittu arkkitehtuurimalli rajapintojen toteuttamiseen. REST on siis kokoelma erilaisia sääntöjä ja rajoituksia, jotka määrittelevät kuinka toteutetaan hypermediajärjestelmä. Noudattamalla näitä rajoituksia saadaan rakennettua arkkitehtuuri, joka on skaalautuva, käytettävä ja saavutettava. REST on siis lähinnä teoreettinen termi ja ohjenuora, eikä mikään standardoitu käsite. [21, s. 2.]

REST:iä käytetään hyvin paljon nykyaikaisten rajapintojen toteutuksissa. Vaikka REST:n säännöt ovat monelle hieman epäselvät, on sen perusperiaate kuitenkin selkeä; resurssit ovat saatavilla verkossa ja jokaiseen resurssiin päästään oman osoitteen kautta.

Tiedonsiirto REST:ssä tapahtuu pääasiassa JSON-formaatissa. JSON (JavaScript Object Notation) on kevytrakenteinen tiedonsiirtoformaatti. Sekä ihmisten, että koneiden on helppo lukea ja sisäistää JSON-muotoisen tiedoston sisältö. JSON ei ole riippuvainen mistään ohjelmointikielestä ja sille löytyy valmiit käsittelykirjastot useimmista ohjelmointikielistä. [22]

3.6.4 Ember CLI

Työssä käytettiin sovelluksen kehittämiseen työkalua nimeltä Ember CLI. Tätä työkalua tarvitaan Ember.js-sovelluksen rungon rakentamiseen, sovelluksen konfigurointiin, pakettien hallintaan, kehittämiseen, testaamiseen ja lopuksi sovelluksen kääntämiseen tuotantovalmiiseen muotoon.

Ember CLI on siis komentokehotteessa toimiva työkalu, jolla saadaan aluksi generoitua ns. “tyhjä sovellus” eli sovelluksen runko. Näin saadaan yhdellä komennolla heti jotain näkyvää ajettavaksi selaimessa ja itse varsinainen kehitystyö voi alkaa nopeasti.

Ember CLI:n avulla konfiguroidaan oma sovellus, eli määritetään lisäosat joita siinä halutaan käyttää. Ember CLI käyttää apunaan kahta eri paketinhallintaa:

Bower, jota käytetään selaimessa käytettävien pakettien ylläpitämiseen ja npm, jolla hallitaan itse sovelluksen sisäisiä paketteja.

Ember CLI:n avulla voidaan myös luoda kaikki tarpeellinen sovellusta varten, esimerkiksi sovelluksen reitit ja mallit. Tällöin Ember CLI huolehtii, että kaikki osat menevät oikeisiin kansioihin ja konfigurointitiedostot päivittyvät samalla.

Ember CLI huolehtii myös kehityksenaikaisen ympäristön ylläpidosta ja sovelluksen automaattisesta kääntämisestä. Automaattinen kääntäminen tarkoittaa sitä, että kun jokin sovelluksen lähdekooditiedostoista muuttuu, niin Ember CLI kääntää automaattisesti uuden version ja päivittää sovelluksen auki olevassa selaimessa. [23]

3.6.5 CoffeeScript

CoffeeScript on yksinkertainen kieli, joka käännetään JavaScriptiksi. CoffeeScriptin avulla voidaan kirjoitettavaa koodia lyhentää huomattavasti, jonka johdosta koodi on myös selkeämpää lukea. CoffeeScriptin tarkoituksena on jättää pois tarpeettomia erikoismerkkejä, kuten aaltosulkeet, puolipisteet ja pilkut. Myös usein toistuvat sanat voidaan jättää kirjoittamatta. CoffeeScriptistä löytyy myös erilaisia vastikkeita ehtolauseille ja loogisille operaattoreille. [24]

CoffeeScript helpottaa huomattavasti koodin lukemista ja kirjoittamista. Suurin osa erikoismerkeistä ja toistuvista sanoista jää pois ja koodi selkeytyy.

4 Sovelluksen suunnittelu

4.1 Vaatimusmäärittely

Moveniumilla oli tarve uudelle sovellukselle, joka olisi nykyaikaisempi kuin käytössä olleet käyttöliittymät. Nämä käyttöliittymät olivat melko normaaleja PHP:lla

rakennettuja web-sovelluksia. Uuden käyttöliittymän piti olla yksinkertainen, helppokäyttöinen ja helposti omaksuttavissa.

Tämä tarkoittaa siis sitä, että tietokanta, itse käyttöliittymän lähdekoodit sekä ohjelmisto olivat kaikki samalla palvelimella ja samassa nipussa. Eli toisin sanoen kaikki näistä olivat vahvasti riippuvaisia toisistaan ja jos yhteen tehtiin muutoksia, niin muitakin täytyi päivittää ajan tasalle. Tarkoitus oli saada hieman modulaarisuutta arkkitehtuuriin.

Yksi suuri lähtökohta oli saada irrotettua sovelluksen käyttöliittymä aivan täysin tietokannasta ja itse sovelluksen logiikasta. Tarkempaa vaatimusmäärittelyä ei tehty, mutta näiden avainasioiden täytyi olla kunnossa:

- Sovellus tulee toimimaan rajapinnan ylitse.
- Rajapinnassa toimii ainoastaan tietokanta- ja sovellusarkkitehtuuri.
- Rajapinta toimii REST-tekniikalla.
- Käyttöliittymä on helppokäyttöinen
- Käyttöliittymä toimii samalla puhelimessa, tabletissa ja tietokoneessa
- Sovelluksen täytyy tukea suomea, ruotsia ja englantia.

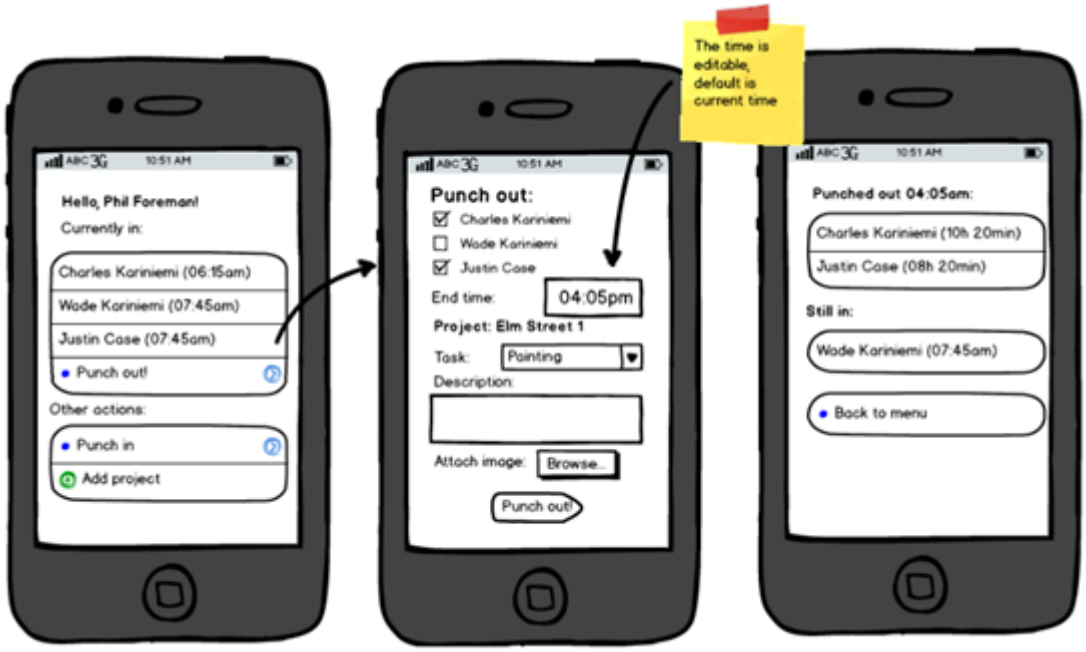
Sovelluksen oli tarkoitus toimia teknologiademona Moveniumille ja jatkossa uudet käyttöliittymät tulisivat pohjautumaan tähän työhön.

Sovelluksesta ei tarvinnut löytyä kuin muutama toiminto; työaikojen lopetus ja aloitus sekä projektien lisääminen. Vaikka sovellus tulisi olemaan yksinkertainen, on itse käytettävyyteen myös panostettava huolella. Tämän työn oli tarkoitus olla myös teknisen demon ohella uusien käyttöliittymätekniikoiden demo.

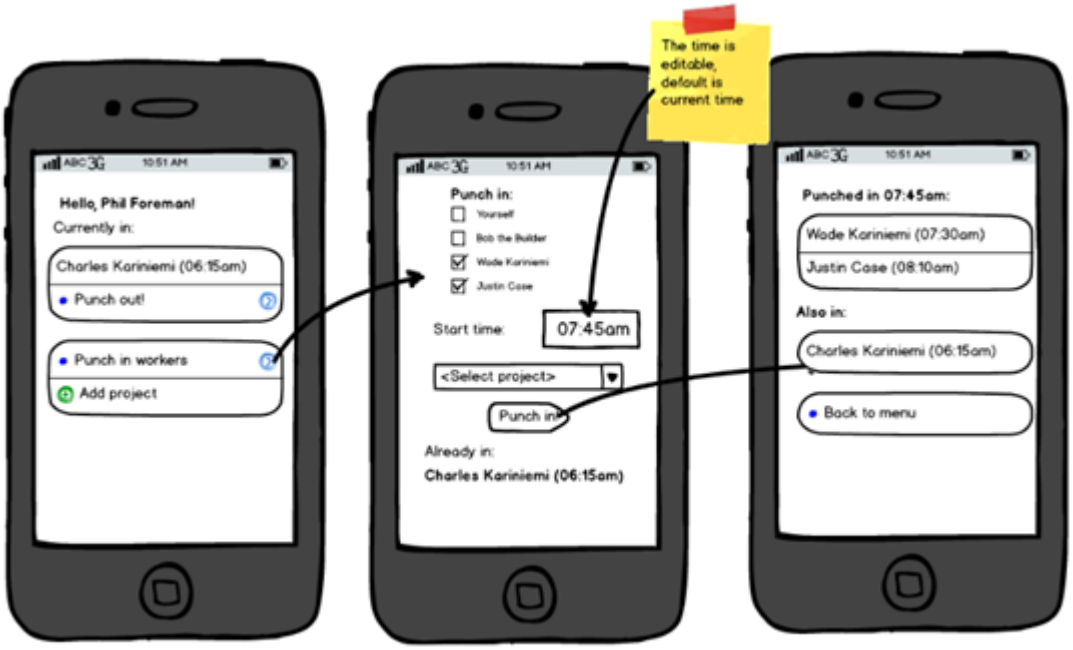
Sovellus oli tarkoitettu asiakasyritysten esimiesten käyttöön ja sitä käytetään pääasiassa älylaitteilla. Tämä käyttäjärooli on siis ns. työnjohtaja, joka toimii yleensä työmaan päällikkönä.

4.2 Käyttöliittymän mallikuvat

Toimeksiantaja toimitti pari mallikuvaa ohjaamaan käyttöliittymän suunnittelua. Kuvissa 1 ja 2 on selkeästi näkyvissä siirtymät eri näkymien välillä ja eri toiminnot.



Kuva 1. Työntekijöiden ulosleimaus



Kuva 2. Työntekijöiden sisäänleimaus

Ulosleimaus tarkoittaa sitä, että esimies valitsee haluamansa työntekijät joilla on avoin työaika ja lopettaa näiden työajan. Sisäänleimaus taas tarkoittaa, että esimies aloittaa valituille työntekijöille työajan.

Aikataulun vuoksi ominaisuudet karsittiin toimeksiantajan kanssa minimiin ja demo-sovellukseen riitti pelkkä etusivulla projektien listaaminen ja poistaminen, sekä tietenkin uusien projektien lisääminen. Tässä työssä toteutettiin siis vain yksinkertainen projektien hallinta mallikuvista poiketen.

4.3 Työkalujen ja menetelmien valinta

Tutkin ja vertailin aikani erilaisia menetelmiä tämän työn toteutukseen. Melkein alusta asti oli selvää, että sovelluksen täytyy olla ns. single-page-sovellus, eli vaikka näkymät ja URL selaimessa vaihtuvat, niin itse sivua ei kuitenkaan ladata kokonaan uudestaan. Tämä nopeuttaa sovelluksen käyttöä huomattavasti ainakin matkapuhelimen verkkoselaimella.

REST-rajapintatekniikasta oli käytännössä jo tehty päätös, koska toimeksiantaja oli aloittanut uuden rajapinnan kehityksen aikaisemmin. Muita rajapintoja ei ollut tarjolla palveluun.

JavaScript-sovelluskehyksistä päällimmäiseksi nousi Ember.js. Tämä oli selkeästi paras työkalu isompien ja raskaiden sovelluksien tekoon. Työssä tehty sovellus tulee luultavasti tulevaisuudessa laajenemaan, joten katsoin Ember.js:n soveltuvan työhön hyvin, vaikka demosovellukseni ei ollutkaan kovin laaja.

Ember.js vaikutti hyvin dokumentoidulta ja olevan helposti omaksuttavissa. Internetistä löytyi myös paljon opastavia kursseja Ember.js:n avulla työskentelyyn. Vaihtoehtoina olleet AngularJS ja Backbone.js olivat nekin kyllä ihan hyviä vaihtoehtoja, mutta Ember.js tuntui soveltuvan tähän työhön ja toimeksiantajan tarpeisiin paremmin.

BootStrap on nykyään melkeinpä standardi, kun puhutaan HTML-komponenttikirjastoista. Se on helppo oppia ja sillä on helppo ja nopea tehdä

yksinkertaisia mutta myös toimivia käyttöliittymiä. HTML-komponenttikirjaston valinta ei tuottanut vaikeuksia.

Ember.js:ää jonkun aikaa opiskeltuani huomasin, että tiedostorakenne ei ollut kovinkaan järkevä eri esimerkeissä. Niissä oli melkeinpä yhteen tiedostoon paketoitu kaikki mahdollinen sovellukseen liittyvä lähdekoodi. Esimerkiksi kaikki Handlebars-templatet kirjoitettiin samaan index.html-tiedostoon, johon tuli muukin HTML-merkkaus. Tämähän ei sinällään haittaa, mutta hankaloittaa kyllä todella paljon suuremman sovelluksen hallinnointia.

Tutkin asiaa hieman ja törmäsin työkaluun nimeltä Yeoman. Yeomanissa oli hienosti ratkaistu tämä ongelma ja muutama muukin. Vaikka Yeoman olikin loistava työkalu, niin Ember.js:n kehittäjät ovat kuitenkin kehittäneet paremman työkalun sovelluksen kehittämiseen; Ember CLI:n.

Versionhallintana toimi Git (ja GitHub), jota toimeksiantaja oli käyttänyt muissakin projekteissa. Git on erittäin toimiva, kun saman projektin parissa työskentelee useampia henkilöitä. Git on monessa mielessä parempi kuin aiemmin käytössä ollut Subversion. Esimerkiksi Gitissä toimii eri haarojen yhdistämiset toisiinsa (merge) loistavasti. GitHubia käytin SourceTree-nimisellä ohjelmalla.

Kehitysympäristönä toimi Windows 7-käyttöjärjestelmä, johon oli asennettu Netbeans IDE. Netbeans on kätevä IDE (integrated development environment) eli integroitu ohjelmointiympäristö, jolla hoituu niin PHP- kuin HTML-projektien ohjelmointi.

Ohjelmistoprojektin eri osioiden suunnitteluun ja seurantaan käytin Asana-nimistä projektinhallintatyökalua.

5 Sovelluksen toteutus ja testaus

5.1 Arkkitehtuuri

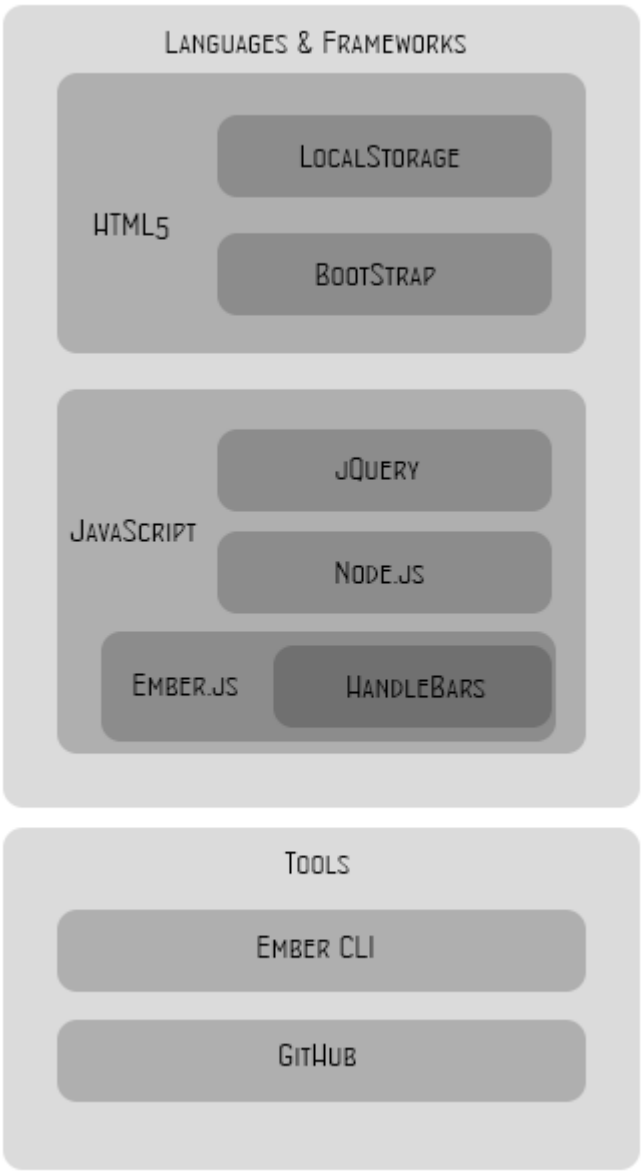
Sovellus oli suoraan MVC-mallin (Model-View-Controller) mukainen, koska Ember.js perustuu vahvasti siihen. Tämä tarkoittaa että nämä kolme eri osiota ovat arkkitehtuurisesti eroteltuina sovelluksen eri osissa.

Model eli malli tarkoittaa järjestelmän tiedon käsittelyä ja tallentamista. Ember.js:ssä on oma Ember Data -luokka, joka huolehtii tästä. Ember Data käyttää myös suoraan REST-rajapintaa erillisen adapterin kautta. Adapteri huolehtii, että tieto liikkuu sovelluksen ja rajapinnan välillä, kun jompaankumpaan tulee muutoksia.

View eli näkymä tarkoittaa asioiden esittämistä käyttäjälle. Tästä huolehtii Ember.js:ssä käytettävät Handlebars-templatetiedostot. Jokaisesta eri sivusta tehdään oma templatetiedosto, jonka avulla ulkoasullisia muutoksia tehdessä ei tarvitse koskea muuhun sovelluksen logiikkaan.

Controller eli käsittelijää hoitaa ohjelmistologiikan mallin ja näkymän välissä. Tästä huolehtii Ember.js:n Controller- ja View-luokat. Näissä osioissa tapahtuu varsinainen tiedon käsittely, eli dataa luetaan mallista ja se käsitellään ja esitellään näkymälle.

Tämä sovellus voidaan myös jakaa eri ohjelmointikielien ja tekniikoiden osalta omiin osioihinsa. Kuva 3 esittää miten koko sovelluksen arkkitehtuurissa eri komponentit ja tekniikat liittyvät toisiinsa.



Kuva 3. Sovelluksen teknologiapuu.

5.2 Kehitysympäristön asennus

Sovelluksen kehitys onnistuu hyvin Windows-ympäristössä, kun asennetaan muutama lisäohjelma. Ember CLI on hyvinkin itsenäinen ohjelma, joka sisältää kaiken tarvittavan Ember.js:llä kehittämiseen.

Ember CLI tarvitsee toimiakseen ensimmäiseksi Noden. Kun tämä on asennettu, voidaan asentaa varsinainen Ember CLI ja sen jälkeen Bower sekä PhantomJS. Selkeät asennusohjeet löytyvät Ember CLI:n dokumentaatiosta. [23]

Kun kaikki tarvittavat komponentit saatiin asennettua, piti Ember CLI-työkalulla luoda ns. runko sovellukselle. Tämä tekee siis kansiorakenteen ja lisäilee joitain pakollisia tiedostoja. Bower-ohjelma huolehtii myös, että eri pakettien riippuvuudet ovat kunnossa. Sovelluksen rungon luonti onnistui komennolla:

```
ember new my-new-app
```

Ember CLI siis loi kansion ja kansion sisällä generoi sovelluksen rakenteen. Kaikkien eri komponenttien asennuksen jälkeen, kehityksenaikainen palvelin saatiin toimimaan komennolla:

```
cd my-new-app
ember server
```

Tämän jälkeen sovellus toimi selaimella osoitteessa http://localhost:4200.

Joskus täytyy päivittää sovelluksen tarvitsemia paketteja ja nämä päivitykset onnistuva komennoilla:

```
npm install
bower install
```

Tarkemmat ja ajantasaiset ohjeet löytyvät Ember CLI:n dokumentaatiosta [25].

5.3 Käyttöliittymä

Käyttöliittymän suunnittelu aloitettiin siitä ajatuksesta, että sen tulisi olla mahdollisimman helppokäyttöinen eikä vaatisi perusteellista kouluttamista. Tästä johtuen sovellukseen toteutettiin vain muutama toiminto ja nekin yksinkertaisia. Vaikka toimintoja tulikin vähän, niin jo tässä vaiheessa täytyi muistaa että sovelluksen toimintojen määrän kasvaessa käyttölogiikka ei saanut huonontua.

Käyttöliittymän suunnittelussa täytyi pitää myös mielessä, että pääasiallinen päätelaite tulisi olemaan kosketusnäytöllinen älypuhelin. Koska älypuhelinten näyttöala on rajallinen, tuli käyttöliittymäkomponentit olla pieniä mutta samalla tietenkin informatiivisia. Napit ja linkit tuli kuitenkin olla sen verran suuria, että niihin osuisi helposti sormella.

Bootstrap tukee suoraan responsiivisuutta, eli käyttöliittymän eri komponentit skaalautuvat näytön vaakaresoluution mukaan. Tämä on erittäin hyödyllistä, kun halutaan että sovellus toimii monella erilaisella päätelaitteella. Tämän sovelluksen oli tarkoitus toimia, näyttää hyvältä ja olla käytettävä älypuhelinten lisäksi tablet-laitteissa ja normaaleilla tietokoneilla.

Käyttöliittymä toteutettiin Bootstrap-kirjastoa hyväksikäyttäen HTML-merkkauskielellä. Ulkoasu pidettiin mahdollisimman yksinkertaisena, joten sivun yläosiossa (header) ei ollut muuta kuin yrityksen ja tuotteen logo sekä linkki uloskirjautumista varten. Sivun alaosiota (footer) ei tehty ollenkaan, koska se ei ollut välttämätön. Alaosiota käytetään yleensä yrityksen tai sovelluksen tietojen esittämiseen.

Taustavärinä on selkeän valkoinen, joka soveltuu älypuhelimiin hyvin. Nappien ja muiden käyttöliittymäkomponenttien värit tulevat Bootstrapin oletuksista. Toimeksiantajalta löytyi valmis brändikonsepti, jossa on värit ja fontit määritelty valmiiksi. Tässä työssä sitä ei kuitenkaan hyödynnetty.

5.4 Näkymät

Sovellukseen tuli siis kolme eri näkymää; etusivu, projektin hallinta ja kirjautumissivu. Kaikki näkymät tehtiin samalla ulkoasulla, jotta sovelluksen käyttö olisi sujuvaa. Etusivulla listataan nykyiset projektit, tämän lisäksi projektin lisääminen ja kirjautumissivu on oma näkymänsä. Projektin muokkaaminen oli käytännössä sama näkyä kuin projektin lisääminen.

Etusivulla listattiin kaikki kyseiselle käyttäjälle kuuluvat projektit. Kyseessähän oli esimies-rooli, joka ei suinkaan näe välttämättä kaikkia yrityksen projekteja, vain ainoastaan hänelle kuuluvat projektit. Näin ollen projekteja oli arvion perusteella maksimissaan parikymmentä, jolloin kaikki projektit voitiin listata samalla sivulla. Jos projekteja olisi useita kymmeniä, jopa satoja, pitäisi listaa jollain tavalla suodattaa. Projektin nimeä klikkaamalla pääsi muokkaamaan projektin tietoja muokkausnäkymään. Projektin nimen kanssa samalla rivillä näkyi ruksi, josta klikkaamalla projektin pystyi poistamaan.

Projektin hallinta, eli lisäys/muokkaus-näkymässä listattiin projektiin kuuluvat tietokentät. Kentät olivat siis suurimmaksi osaksi teksti- ja numerokenttiä, joihin tallennettiin erinäistä tietoa projektista. Osa kentistä voi olla pakollisia. Näkymän alaosassa oli Tallenna-nappi ja Peruuta-nappi, josta pääsi pois näkymästä takaisin etusivulle. Tallenna-nappi joko lisää uuden projektin tai muokkaa vanhaa, riippuen ollaanko lisäys- vai muokkausnäkymässä.

Kirjautumisnäkymä sisälsi kaksi kenttää, jotka olivat käyttäjätunnus ja salasana. Käyttäjä syötti niihin tunnuksensa, jotka ovat samat kuin Movenium-tunnukset. Näkymässä ilmoitettiin kun kirjautuminen oli käynnissä ja jos kirjautumisessa tapahtui jokin virhe. Kun kirjautuminen oli hyväksytty, siirrettiin käyttäjä etusivulle.

5.5 Tietoturva

Sovellukseen tarvitaan tunnukset, jotta sitä voitaisiin käyttää. Tunnuksiksi käy samat, joita käytetään yleisesti Moveniumin tietojärjestelmiin. Opinnäytetyötä varten sain tarvittavat tunnukset toimeksiantajalta.

Sovelluksessa käytetään OAuth-protokollan [26] mukaista autentikointia ja autorisointia. Autentikointi tarkoittaa sitä, että kun käyttäjä kirjautuu käyttäjätunnuksella ja salasanalla, niin REST-rajapinta varmistaa onko käyttäjä oikea. Jos käyttäjä on todennettu oikeaksi, niin rajapinta palauttaa tälle avaimen (access token). Access token on turva-avain, joka on sidottu käyttäjään. Avain on voimassa vain tunnin kerrallaan, jonka jälkeen se täytyy uusia. Autentikoinnissa

tulee vastauksena myös “refresh token”. Tämä on avain, jolla saadaan haettua uusi avain ilman käyttäjätunnusta ja salasanaa.

Autorisoinnilla tarkoitetaan, että jokainen kutsu rajapintaan varmennetaan autentikoitumisella saadulla avaimella. Tällä tavoin rajapinta myös tietää kenestä käyttäjästä on kyse, eikä informaatiota näytetä sellaisille käyttäjille joille se ei kuulu. Sovelluksessa jokainen näkymä täytyy olla autorisoinnin takana, jotta niitä ei voi nähdä ennen kuin rajapinta on varmentunut käyttäjästä.

TimeTrackerin REST-rajapinta huolehtii käytännössä kaikesta tietoturvaan liittyvästä. Rajapinnan tehtävänä on huolehtia siitä, mitkä resurssit kuuluvat kenellekin. Kaikki HTTP-liikenne on myös salattu SSL-tekniikalla. Rajapinta huolehtii myös, että esimerkiksi lomakkeilla ei voi syöttää vääränlaista tietoa.

Sovelluksessa käytettiin *ember-simple-auth* -lisäosaa, joka huolehtii kirjautumisesta ja sen säilyvyydestä. Lisäosan avulla voidaan hakea aina uusi access token automaattisesti, ennen kuin se vanhentuu. Lisäosa pitää myös kirjautumisen muistissa selaimen LocalStoragessa, jolloin jokaisella kerralla ei tarvitse kirjautua sovellukseen uudestaan käyttäjätunnuksella ja salasanalla. [27]

5.6 Lokalisointi

Sovelluksen täytyi toimia suomeksi, ruotsiksi ja englanniksi, koska toimeksiantajan asiakkaina on käyttäjiä eri maista. Eri kieliversioiden lisäksi täytyy lokalisoida myös muita tietoja, esim. päivämääriä ja valuuttoja.

Kieliversiot tehtiin eri tiedostoihin JSON-muotoisena. Yksi kieli on siis yksi tiedosto, esimerkiksi *fi.json*. Käyttäjä voi vaihtaa kielen helposti ilman, että koko sivua tarvitsee ladata uudelleen, riittää että uusi kielitiedosto ladataan.

Päivämäärien ja kellonaikojen lokalisointi toteutettiin Moment.js-kirjastolla. [28]

5.7 Mallien luonti dynaamisesti

Normaalisti mallit luodaan kiinteästi Ember.js-sovellukseen ja ne määritellään omiin tiedostoihin. Tämä on aivan normaali käytäntö, koska yleensä sovelluksen arkkitehtuuri on etukäteen tiedossa.

Movenium TimeTracker -palvelu on asiakaskohtaisesti määriteltävissä hyvinkin tarkasti ja ei voida etukäteen tietää, mitä kenttiä esimerkiksi projektit-mallissa on. Asiakkaat voivat myös itsekin määritellä omaa palveluaan.

Tästä johtuen mallit täytyy luoda dynaamisesti, kun sovellusta alustetaan. Tämä tapahtuu niin, että rajapinnasta kutsutaan heti kirjautumiseen jälkeen:

```
GET /forms
```

Tämä kutsu palauttaa listauksen lomakkeista ja niiden kentistä ja asetuksista, jotka kuuluvat kyseiseen palveluun. Kutsussa tuleva listaus käydään iteroimalla läpi ja jokaisesta lomakkeesta tehdään oma malli ja jokaisesta lomakkeen kentästä tehdään mallille attribuutti.

5.8 REST-adapterin käyttö

Ember.js sovellus käyttää aina jotain adapteria datan tallentamiseen tietovarastoon. Adapteri siis määrittelee, miten sovelluksessa käsiteltävä data tallennetaan palvelimelle tai lokaalisti ja miten sitä haetaan sieltä. Tässä sovelluksessa backendin virkaa hoitaa REST-rajapinta, joten pitää käyttää adapteria joka ymmärtää sitä.

Ember.js:stä löytyy itsestään valmiiksi toteutettu REST-adapteri, mutta joitakin eroavaisuuksia löytyy palvelimen REST-rajapinnan käyttäytymisestä ja Ember.js:n olettamuksista, joten joitakin muutoksia siihen täytyi tehdä. Adapterille määriteltiin rajapinnan URL-osoite ja polku, sekä jokaiseen palvelinkutsuun lisättiin aina OAuth-token.

5.9 Movenium Collector

Movenium Collector on nimitys Moveniumin käyttämälle ja kehittämälle sovelluksen arkkitehtuurille. Collector käsittää tietokannan, REST-rajapinnan ja muutamia muita komponentteja, joihin en ottanut tässä työssä kantaa.

5.9.1 Movenium REST API -rajapinta

Collectorin rajapintaa kutsutaan nimellä Movenium REST API. Rajapinnan avulla voidaan hakea tietoa ja käsitellä sitä Moveniumin palveluissa. Rajapinnan käyttöön vaaditaan tunnukset sekä palvelusopimus Moveniumin kanssa.

Rajapintaa käytetään niin, että ensin kirjaudutaan sisään jolloin saadaan OAuthin tarjoama token. Tällä tokenilla voidaan käyttää kaikkia niitä resursseja, joihin kyseisellä käyttäjätunnuksella on pääsy.

5.9.2 Järjestelmän tietorakenne

Movenium Collector -järjestelmä eroaa tietorakenteeltaan jonkin verran perinteisestä ohjelmistosta. Asiakkaiden palvelut voivat olla hyvinkin erilaisia ja dynaamisia, joten tämä asettaa erityisiä vaatimuksia tietokannan rakenteeseen.

Perinteisessä relaatiotietokannassa käyttäjät ovat omassa taulussaan ja projektit omassaan. Mukana on aputaulu, jossa käyttäjät ja projektit linkitetään toisiinsa. Näin tiedetään, mikä projekti on linkitetty millekin käyttäjälle. Käyttäjät-taulussa on tietty määrä erilaisia sarakkeita ja samalla tavalla projektit-taulussa on eri sarakkeita, joissa on erityyppisiä tietoja.

Tämä rakenne ei toimi järkevästi, jos osa asiakkaista käyttää projektit-taulussa aivan eri tietoja kuin toiset asiakkaat. Yksi asiakas saattaa esimerkiksi haluta

uuden kentän projektit-tauluun ja perinteisessä relaatiotietokannassa se pitäisi aina lisätä käsin tietokannan tauluun uutena kenttänä.

Collectorissa asia on ratkaistu niin, että data on hajautettu eri tietokannan tauluihin. Asiakkaat ja niiden palvelut on määritelty erillisten konfiguraatiotiedostojen avulla. Tietokanta onkin ns. virtuaalinen tietokanta, joka on rakennettu MySQL-tietokannan päälle. Collector hoitaa kaiken datan käsittelyn omilla luokillaan ja metodeillaan. [29]

5.10 Sovelluksen testaus

Testaamisen avulla etsitään sovelluksesta virheitä ja muita epäkohtia, esimerkiksi epäloogisuuksia käyttölogiikassa. Sovelluksen kehittäjä ei välttämättä itse huomaa kaikkia epäkohtia, joten on järkevää suorittaa testausta niin, että muita henkilöitä on siinä osallisena.

Sovellusta pitää aina testata ennen varsinaista julkaisua tuotantokäyttöön. Testauksen avulla saadaan enimmät virheet selville ja niitä voidaan vielä korjata. Ohjelmiston testaustapoja on olemassa monia erilaisia ja tässä työssä käytin muutamaa erilaista testaustapaa.

Testaamista tapahtuu tietenkin koko ajan myös kehityksen aikana, mutta varsinainen lopputestaus on aina tehtävä erikseen. Tässä työssä Ember CLI hoiti kehityksenaikaista testaamista yksikkö- ja integraatiotestien avulla.

5.10.1 Yksikkötestaus

Kun halutaan testata sovelluksen eri osioiden toimivuutta kehitystyön aikana, on yksikkötestaus oiva työkalu. Jokainen testitapaus on kirjoitettava erikseen ja yleensä itse testitapaus kirjoitetaan ensin ja sitten vastaava sovelluksen komponentti, joka toimii automaattisesti halutulla tavalla.

Tällä tavalla mietitään jo etukäteen miten kyseisen komponentin halutaan toimivan ja päädytään yleensä parempaan ohjelmiston rakenteeseen. Usein yksikkötestit kuitenkin kirjoitetaan jälkikäteen, jolloin ne eivät ehkä ajatustasolla ole niin tehokkaita.

Jälkikäteen tehdyt sovelluksen toimintatapojen muutokset vaativat myös muutokset testitapauksiin ja tämä lisää kehittäjien työtä. Jos testitapausta ei päivitetä, se yleensä johtaa epäonnistuneeseen testiin ja käy helposti niin, että koko testitapaus otetaan pois päältä. [30]

Tässä työssä käytin Ember QUnit -työkalua [31], joka tulee Ember.js:n mukana. Kirjoitin testit jokaiselle sovelluksen toiminnolle, eli projektien hakemiselle, lisäämiselle, muokkaamiselle ja poistamiselle.

5.10.2 Integraatiotestaus

Integraatiotestauksen avulla testataan sovelluksen eri komponenttien vuorovaikutusta toisiinsa. Kun yksikkötestauksen testitapaukset on tehty, voidaan tehdä erinäisiä testitapauksia komponenttien välisiin suhteisiin integraatiotestauksen avulla. Usein integraatiotestauksessa esiin tulevat virheet johtuvan enemmänkin suunnittelun epäkohdista, kuin varsinaisista ohjelmointivirheistä. [32]

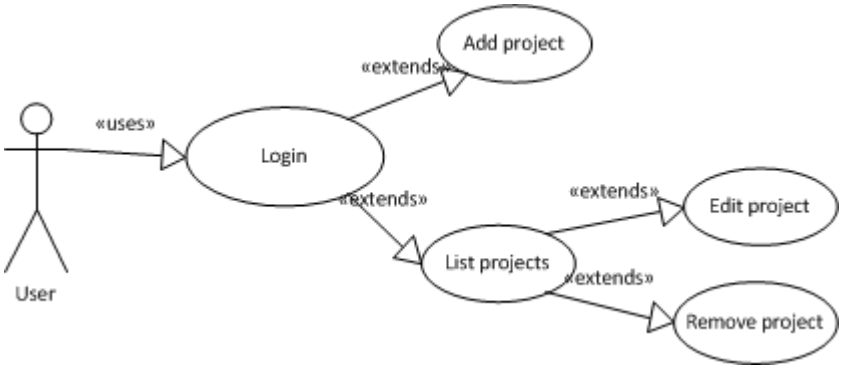
Tässä työssä oli kuitenkin niin vähän komponentteja, että en nähnyt tarvetta varsinaiselle integraatiotestaukselle, vaikka Ember QUnit -työkalulla siihen jonkinlainen tuki olisi löytynytkin. [31]

5.10.3 Käyttötapausanalyysi

Yksi helppo tapa testata oletettavasti valmista sovellusta on käyttää testitapausten määrittämiseen käyttötapauksia. Otetaan pohjalle sovelluksen käyttötapaukset ja näistä määritellään muutamia testitapauksia jokaista käyttötapausta kohden. Tarkoitus on saada siis testattua esimerkiksi kelvolliset ja ei-kelvolliset

syötteet lomakkeissa. Tässäkin testitavassa on tarkoituksena löytää mahdollisimman paljon vielä virheitä. [33]

Käytin ulkopuolisia henkilöitä käyttötapausten (kuva 4) avulla testaamiseen ja samalla keräsin tietoa sovelluksen käyttökokemuksesta yleisesti analysointia varten. Testitapaukset ja lisäkysymykset näkyvät taulukossa 1. Testien lopuksi oli mahdollisuus antaa palautetta sovelluksesta ja kirjata jokaisen testitapauksen havainnot.



Kuva 4. Sovelluksen käyttötapaukset

Taulukko 1. Käyttötapauksista luodut testitapaukset (teksteistä on poistettu Movenium Oy:n palveluiden osoitteita ja käyttäjätunnuksia)

Testi-tapaus Nro	Käyttötapaus	Kuvaus	Vaiheet	Odotettu tulos
1	Login	Avaa sovellus verkkoselaimella ja kirjaudu sisään.	1. Avaa selain osoitteeseen [palvelun osoite] 2. Kirjaudu sisään, käyttäjätunnus on [tunnus] ja salasana [salasana]	Kirjaudutaan käyttäjänä palveluun sisään
2	Add project	Lisää projekti	1. Lisää projekti 2. Anna sille nimeksi "Testi" 3. Voit halutessasi syöttää tietoa muihinkin kenttiin	Projekti lisätään palveluun
3	List projects	Listaa projektit	1. Mene "Kohteet"-sivulle	Kohdassa 2. luotu projekti pitäisi näkyä listassa
4	Edit project	Muokkaa projektia	1. Muokkaa kohdassa 2 luomaasi projektia 2. Anna sille nimeksi "Testi muokattu" 3. Voit myös muokata muitakin kenttiä	Projektin tiedot muuttuvat
5	Remove project	Poista projekti	1. Poista kohdassa 2 luomasi projekti	Projekti poistetaan

5.10.4 Hyväksymistestaus

Viimeisen testauksen ideana on käydä sovellus kokonaan läpi käyttöliittymän osalta ja varmistaa, että virheitä ei enää löydy. Hyväksymistestauksen tekee mieluiten sovelluksen tilaaja, se taho kenelle sovellus luovutetaan.

Testauksen aikana katsotaan vastaako sovellus määriteltyjä ominaisuuksia ja toimiiko se oikein. Hyväksymistestauksen aikana ei ole enää tarkoitus etsiä mahdollisia virheitä, vaan varmistaa, että sovellus toimii halutulla tavalla. [34]

Tässä työssä suoritimme hyväksymistestauksen yhdessä toimeksiantajan kanssa.

5.11 Sovelluksen julkaisu

Kun testaukset on suoritettu hyväksytysti ja on todettu, että sovellus on vaatimusmäärittelyn mukainen, voidaan se julkaista tuotantokäyttöön. Ennen julkaisua on kuitenkin selvitettävä palvelimen vaatimuksia ja miten sovellus jaellaan sinne.

5.11.1 Palvelimen vaatimukset

Työn mukainen JavaScript-sovellus ei vaadi juuri paljoakaan suoritustehoa palvelimelta, koska osa laskennasta tapahtuu käyttäjän selaimessa JavaScript-sovelluskehyksen ominaisuuksien takia. REST-rajapinta on myös erillinen komponentti, joka toimii omassa palvelinympäristössään. Rajapinnan palvelinta ei käsitelty tässä työssä. Sovelluksen ei tarvitse siis itsessään suorittaa tietokantahakuja tai muitakaan raskaita prosesseja ollenkaan. Suurin osa datan manipulaatiosta tehdään rajapinnassa, joka keventää sovelluksen kuormaa entisestään.

Sovelluksen palvelimen tehtäväksi jääkin siis tarjota palvelintila sovelluksen käyttämille tiedostoille ja jaella niitä Http-palvelun (esim. Apache tai Nginx) avulla. Palvelimen pitää tietenkin pystyä käsittelemään jokainen Http-kutsu ilman, että siitä aiheutuu merkittäviä viiveitä. Voidaan siis olettaa, että palvelimeksi riittää samanlainen palvelin kuin normaaleille web-sivustoille.

Näin ollen ei voida suoraan sanoa, että mitä itse Ember.js-sovellus vaatii palvelimelta. Ensin pitää selvittää miten paljon ylipäätään liikennettä on tulossa palvelimeen, jonne sovellus julkaistaan ja sen mukaan määrittää palvelimen tehovaatimuksia. Tämän työn sovellusta ei tullut käyttämään kovinkaan moni ihminen, koska tämä oli teknologiademo, joten palvelimelle ei ollut erityisiä vaatimuksia.

Jos haluaa vielä jakaa palvelimen kuormaa, voidaan staattiset tiedostot jaella CDN-palvelun (Content delivery network) kautta. Staattisia tiedostoja ovat esimerkiksi kuva- ja fonttitiedostot.

5.11.2 Sovelluksen paketointi ja julkaisu

Sovellus täytyy ensin paketoida sopivaan muotoon, jolloin se voidaan ladata palvelimelle. Ember CLI sisältää työkalut sovelluksen paketointia varten.

Ember CLI pakkaa JavaScript- ja CSS-tiedostot pienempään muotoon, jolloin ne vievät vähemmän kaistaa ja latautuvat nopeammin. Myös kuvat ja esimerkiksi fontit voidaan tässä vaiheessa pakata pienempään muotoon, mutta se ei ole välttämätöntä.

Julkaisu tehdään yksinkertaisesti niin, että paketoidut tiedostot vaan ladataan julkiselle verkkopalvelimelle. Tässä työssä käytin toimeksiantajan testipalvelinta, jonne sovellus asennettiin toimimaan.

5.11.3 Loppukäyttäjien ohjeistus

Koska sovelluksen suunnitteluperiaatteena oli, että se olisi mahdollisimman helppokäyttöinen, ei tarvita erillistä ohjedokumenttia käyttäjiä varten. Tein kuitenkin sovelluksen sisään yhden näkymän, jossa on listattu yleisimpiä sovelluksen käyttöön liittyviä ohjeita.

Jos sovelluksesta tehdään laajempi, myös ohjeita täytyy laajentaa sen mukaan. Sovelluksen sisäinen ohjeistus on johonkin tasoon asti riittävä, mutta jossain vaiheessa täytyy tehdä erillinen ohjedokumentti. Kun taas sovellus kasvaa edelleen ja erilaisia toimintoja ja toimintatapoja tulee lisää, niin erillinen dokumenttikaan ei välttämättä enää riitä ja asiakkaille joudutaan järjestämään koulutustilaisuuksia tarpeen vaatiessa.

6 Työn tulokset

6.1 Ember.js- ja PHP-sovellusten vertailu

Vertailua varten käytin Movenium TimeTracker -palvelun vanhempaa versiota. Vanhempi versio on toteutettu PHP:lla ja voidaan näin ollen sanoa sitä ns. perinteiseksi web-sovellukseksi. Tällaisessa perinteisessä web-sovelluksessa koko sivunäkymä ladataan aina joka kerta uudestaan. Tässä testissä kutsun uutta Ember.js-sovellusta TT4:ksi ja vanhaa PHP-sovellusta TT3:ksi.

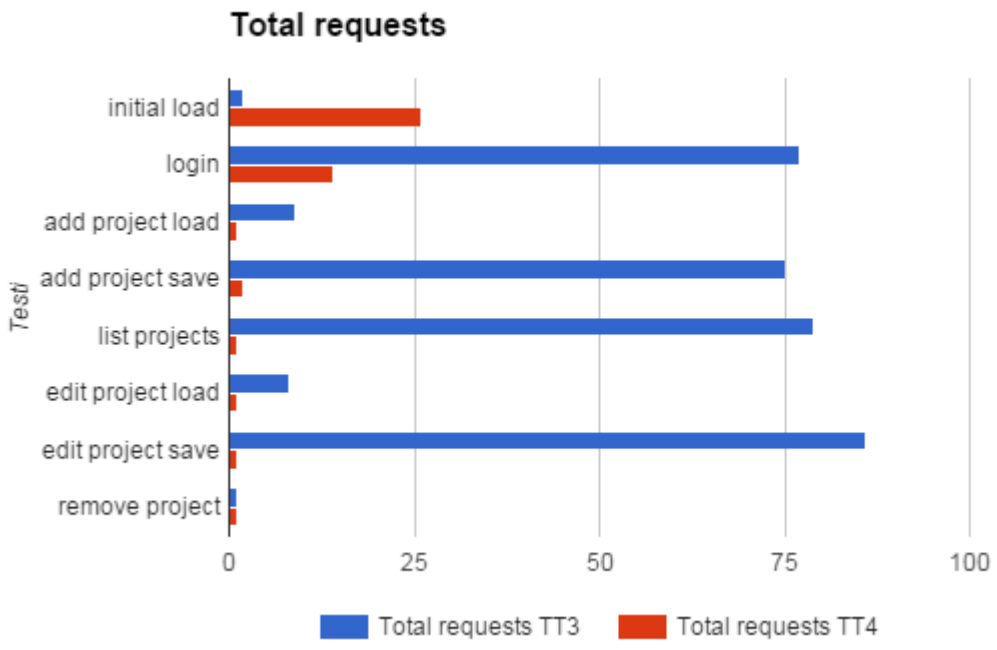
Pienen käyttöjakson jälkeen pystyin arvioimaan käyttökokemuksen perusteella, että TT4 tuntuu nopeammalta kuin TT3. Tämä johtuu siitä, että resursseja ladataan asynkronisesti monia samaan aikaan eikä jokainen toiminto vaadi sivun lataamista uudelleen. Sovelluksen ensilataaminen on kuitenkin TT4:ssa hitaampaa kuin TT3:ssa, koska sovelluksen käyttämät JavaScript-tiedostot ovat melko suuria. TT3:ssa ei varsinaista ensilataamista tapahdu, koska kaikki laskenta tapahtuu palvelimella, eikä loppukäyttäjän selaimessa, kuten TT4:ssa.

Tein käyttötapauksiin pohjautuvia nopeustestejä eri sovellusten välillä. Testaamiseen käytin Chrome-selainta, jossa oli päällä nopeuden rajoittaminen nopeuteen 750 kbps. Nopeuden rajoittaminen helpottaa sovellusten vertailua, koska tällä tavalla erot kasvavat. Selaimesta on myös välimuisti pois käytössä.

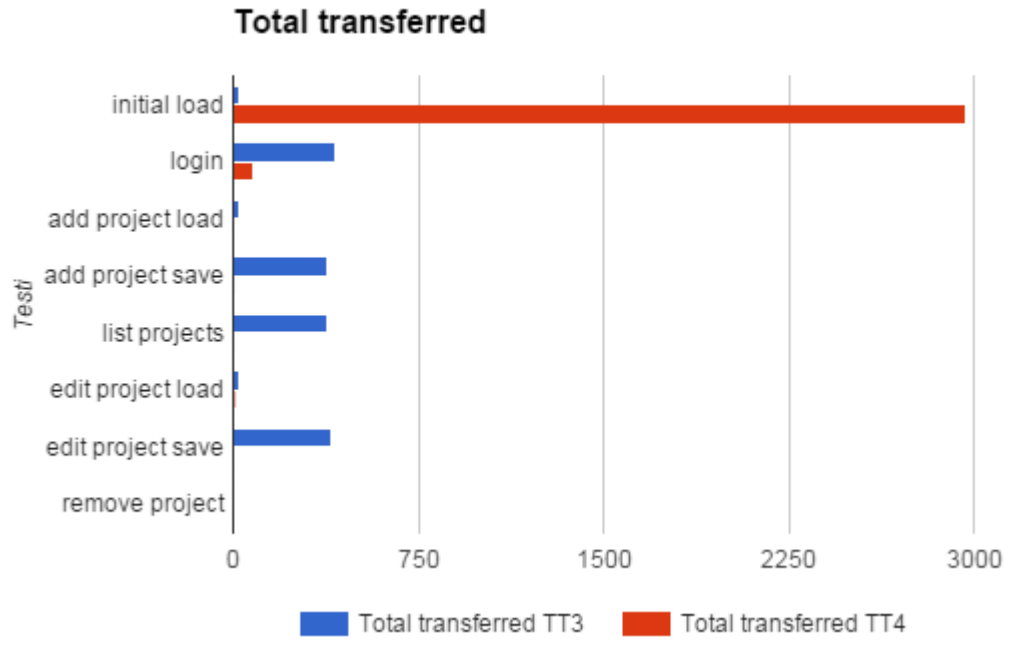
Testeissä mittasin kolmea eri suuretta: kuinka monta kutsua tehtiin palvelimelle, kuinka paljon dataa siirrettiin ja kuinka pitkään lataus kesti. Testien tulokset ja niistä tehdyt kuvaajat näkyvät alla (taulukko 2, kuvat 5-7).

Taulukko 2. Nopeustestien tulokset.

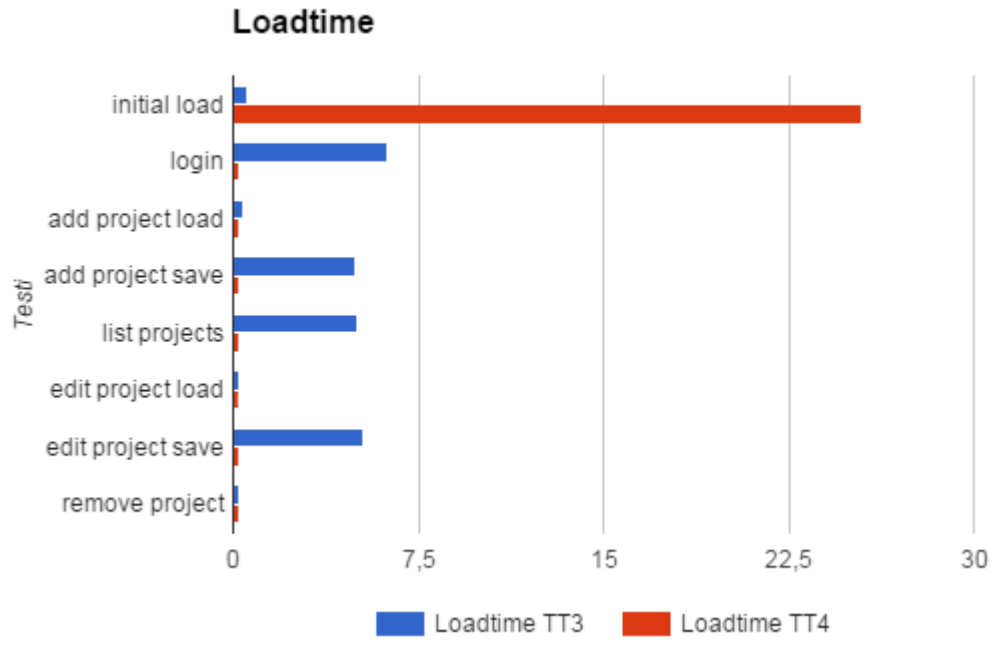
Testi	Total requests TT3	Total requests TT4	Total transferred TT3 (KB)	Total transferred TT4 (KB)	Loadtime TT3 (s)	Loadtime TT4 (s)
initial load	2	26	21	2969	0,595	25,45
login	77	14	417	82,6	6,23	0,201
add project load	9	1	21	0,333	0,395	0,207
add project save	75	2	380	1,3	4,98	0,237
list projects	79	1	383	0,63	5,01	0,245
edit project load	8	1	20	7	0,25	0,188
edit project save	86	1	401	0,47	5,24	0,237
remove project	1	1	0,45	0,3	0,236	0,254
	337	**47**	**1643,45**	**3061,633**	**22,936**	**27,019**



Kuva 5. Palvelinkutsujen määrä.



Kuva 6. Datasiirron määrä.



Kuva 7. Latausnopeus.

Nopeustestit näyttävät, että TT4:n ensimmäinen lataaminen kyllä kestää kauemmin mutta itse sovelluksen käyttäminen on nopeampaa kuin TT3:ssa. Voidaan siis todeta, että uusi tekniikka nopeuttaa sovelluksen käyttöä loppukäyttäjän näkökulmasta. Koska testissä nopeus oli rajoitettu, niin normaalitilanteessa ensilataus toimisi huomattavasti nopeammin.

Palvelimelle tehdyt pyynnöt vähenevät myös uudella tekniikalla ja ne ovat huomattavasti pienempiä kooltaan. Vaikka testien summissa TT4:ssa on hieman enemmän siirrettyä dataa, niin täytyy muistaa, että ensilatauksen osuus oli n. 97 %. Uusi tekniikka siis kuormittaa palvelimia vähemmän.

TT3:ssa on projektin poisto tehty Ajax-tekniikalla, jonka johdosta koko sivua ei tarvitse uudestaan ladata, eikä turhia pyyntöjä tule palvelimelle.

6.2 Käyttötapausanalyysin tulokset

Käytin kahta eri henkilöä käyttötapausanalyysillä testaamiseen. Pääosin testit menivät läpi ongelmitta, eikä kriittisiä virheitä löytynyt. Määritellyt käyttötapaukset toimivat siis suunnitellusti.

Jotkin lisätoiminnot eivät toimineet halutulla tavalla, esimerkiksi kuvaan lisääminen projektille ei onnistunut. Projektin lisätietojen syöttäminen vääränlaisella formaatilla meni myös joidenkin tietojen osalta läpi. Nämä virhetilanteet kuitenkin saatiin käyttötapausanalyysin perusteella selville ja ne voidaan myöhemmin korjata.

Käyttötapausanalyysin tulosten perusteella sovellukseen ei tarvitse tehdä muutoksia.

6.3 Testihenkilöiden palaute

Ensimmäinen testihenkilö ei päässyt läpi toista käyttötapausta, koska ohjedokumentissa ja sovelluksessa oli erilaiset termit. Sovelluksessa on aiemmin käy-

tössä ollut termi “projekti”, joka on myös käyttötapausanalyysissa. Nykyään sovelluksessa käytetään termiä “kohde”.

Toinen testihenkilö antoi palautetta eri syötteiden validoinnista. Nykyisessä sovellusversiossa erityyppiset syötteet tarkistetaan, mutta jatkokehitystä näille luultavasti vielä tarvitaan. Kuitenkin REST-rajapinta tekee vielä omat tarkistukset käyttäjän syötteille, joten mitään kriittistä virhettä ei kuitenkaan pääse syntymään.

6.4 Sovelluksen jatkokehitys

Tässä työssä saatiin todistettua, että uusi JavaScript-sovelluskehystekniikka soveltuu hyvin toimeksiantajan tarpeisiin. Tällä saadaan parannettua loppukäyttäjien käyttökokemusta, tehtyä sovelluksesta modulaarisempia ja kevennettyä palvelimen kuormaa.

Työssä tehtyä esimerkkisovellusta käytettiin lähestulkoon sellaisenaan uuden sovellusversion alustaksi. Sovellusta on nyt jatkokehitetty Moveniumilla pari vuotta hyvinkin onnistuneesti. Tällä hetkellä (2016) uutta sovellusversiota myydään pääasiallisena versiona ja vanhempi versio on jäädytetty ohjelmistokehityksen osalta.

Jatkossa sovelluksesta kehitetään vieläkin modulaarisempaa ja enemmän Ember.js:n toimintalogiikan mukaiseksi. Joitain sovelluksen komponentteja joudutaan kirjoittamaan uudelleen, koska uudet Ember.js-versiot ovat tuoneet mukanaan useita parannuksia arkkitehtuuriin.

7 Loppupohdinta

Opinnäytetyö oli toiminnallinen työ, jossa toimeksiantajalta tuli selkeät vaatimusmäärittelyt. Näin ollen oli helppo suunnitella millainen sovelluksen piti olla. Työn tarkoituksena oli tutkia uusia menetelmiä web-sovellusten kehittämiseen

ja miten hyvin ne soveltuvat toimeksiantajan käyttöön. Työn toteuttamisessa ei ollut ongelmia ja sovellus vastasi hyvin vaatimusmäärittelyä.

Opinnäytetyön tekeminen oli pitkä prosessi; se aloitettiin syksyllä 2013 ja valmistui keväällä 2016. Itse työhön liittyvän sovelluksen kehittäminen oli melko nopea vaihe heti aluksi, se kesti vain pari kuukautta. Valmiin sovelluksen pohjalta toimeksiantaja alkoi kehittää itselleen varsinaista sovellusta, jossa hyödynnettiin tämän työn tuloksia. Tällä hetkellä uusi sovellusversio alkaa syrjäyttää vanhempaa versiota ja kaikki uusi ohjelmistokehitys tapahtuu vain uuteen sovellukseen.

Itse opin työn aikana paljon JavaScript-sovelluskehysten, erityisesti Ember.js:n, käyttämisestä ohjelmistojen kehittämiseen. Minulla ei ollut aiemmin juuri lainkaan kokemusta tämäntyyppisistä työkaluista, olin kehittänyt ohjelmistoja monta vuotta melkein ainoastaan PHP:lla. Myös REST-rajapintamalli on osoittautunut tärkeäksi työkaluksi ja työn avulla opin siitäkin paljon.

Lähteet

1. Gartner. Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013. [Viitattu 14.2.2014] Saatavissa: http://www.gartner.com/newsroom/id/2665715
2. W3Schools.com. OS Platform statistics 2014. [Viitattu 14.2.2014] Saatavissa: http://www.w3schools.com/browsers/browsers_os.asp
3. Nations, D. What is a Web Application [Viitattu 17.2.2016] Saatavissa: http://webtrends.about.com/od/webapplications/a/web_application.htm
4. html5test.com. How well does your browser support HTML5? [Viitattu 8.3.2014] Saatavissa http://html5test.com/results/mobile.html
5. The Apache Software Foundation. Apache Cordova. [Viitattu 10.12.2015] Saatavissa: https://cordova.apache.org/
6. Työaikalaki 605/1996
7. Kotimaisten kielten keskus. Lyhenneluettelo. [Viitattu: 12.2.2013] Saatavissa: http://www.kotus.fi/index.phtml?s=2149
8. Movenium Oy. Movenium TimeTracker työajanseuranta. [Viitattu: 12.2.2014] Saatavissa: http://movenium.com/home/fi/palvelut/timetracker_työajanseuranta
9. Mozilla Developer Network. JavaScript. [Viitattu 14.2.2014] Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript
10. W3C. A short history of JavaScript. [Viitattu 8.3.2014] Saatavissa https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript
11. Harrell, W. HTML, CSS & JavaScript Mobile Development For Dummies. 2011.

12. Gross, C. Ajax and REST Recipes: A Problem-Solution Approach. 2006.
13. PHP Documentation Group. PHP Manual. [Viitattu 9.3.2014] Saatavissa http://www.php.net/manual/en/index.php
14. Vaswani, V. How to Do Everything with PHP & MySQL. 2005.
15. Bodmer, M. Ember.js Application Developement How-to. 2013.
16. Google. What Is Angular? AngularJS: Developer Guide: Introduction. [Viitattu 10.3.2014] Saatavissa: http://docs.angularjs.org/guide/introduction
17. Genev, M. Backbone or Angular or Ember? [Viitattu 10.3.2014] Saatavissa: http://www.100percentjs.com/backbone-or-angular-or-ember-here-is-my-choice-and-why
18. W3Techs. Usage Statistics and Market Share of JavaScript Libraries for Websites, March 2014. [Viitattu 10.3.2014] Saatavissa: http://w3techs.com/technologies/overview/javascript_library/all
19. The jQuery Foundation. What is jQuery? [Viitattu 10.3.2014] Saatavissa: http://jquery.com/
20. Bootstrap [Viitattu 10.3.2014] Saatavissa: http://getbootstrap.com/
21. Wilde, E. Pautasso, C. REST: From Research To Practise. 2011.
22. IETF. The application/json Media Type for JavaScript Object Notation (JSON) [Viitattu 25.2.2016] Saatavissa: http://www.ietf.org/rfc/rfc4627.txt
23. Tilde Inc. Ember CLI. [Viitattu 10.12.2014] Saatavissa: http://ember-cli.com/
24. CoffeeScript [Viitattu 10.12.2015] Saatavissa: http://coffeescript.org/
25. Tilde Inc. Ember CLI user guide [Viitattu 10.12.2015] Saatavissa: http://ember-cli.com/user-guide/
26. OAuth Community Site [Viitattu 10.12.2015] Saatavissa: http://oauth.net/
27. simplabs GmbH/Marco Otte-Witte. simplabs/ember-simple-auth [Viitattu 10.12.2015] Saatavissa: https://github.com/simplabs/ember-simple-auth
28. Moment.js [Viitattu 10.12.2015] Saatavissa: http://momentjs.com/
29. Salminen, V, Virtuaalinen Tietokanta Sovellusalustana. 2014.
30. Xie, T. Towards a Framework for Differential Unit Testing of Object-Oriented Programs [Viitattu 18.12.2015] Saatavissa: http://people.engr.ncsu.edu/txie/publications/ast07-diffut.pdf
31. Florence, R. Ember QUnit [Viitattu 18.12.2015] Saatavissa: https://github.com/rwjblue/ember-qunit
32. Martyn, O & Charles, U. Testing in Software Development, s71. 1986.
33. Jormanainen, N. Käyttötapauspohjainen testaaminen. 2006.
34. ISTQB. What is Acceptance testing [Viitattu 19.12.2015] Saatavissa: http://istqbexamcertification.com/what-is-acceptance-testing/
35. Crockford, D. JavaScript: The World's Most Misunderstood Programming Language [Viitattu 2.3.2016] Saatavissa: http://www.crockford.com/javascript/javascript.html